M-8637-1P US
816441 v2

EXPRESS MAIL LABEL NO:

EL901564855US

SYSTEM AND METHOD FOR CONNECTING A LOGIC CIRCUIT SIMULATION TO A NETWORK

Robert M. Zeidman

CROSS-REFERENCE TO RELATED PATENT APPLICATION

The present application is a continuation-in-part application of copending U.S. patent application, entitled “METHOD FOR CONNECTING A HARDWARE EMULATOR TO A NETWORK,” serial no. 09/751,573, filed on December 28, 2000.

BACKGROUND OF THE INVENTION

Prior to reducing an integrated circuit design to a form suitable for fabrication, the integrated circuit design is often simulated in software on a computer, and emulated in hardware to allow the design to be optimized and debugged. Typically, using a hardware description language (e.g., VHDL), the circuit designer prepares a hardware description of the integrated circuit, which is then compiled into a software model to be simulated on a computer (e.g., an engineering workstation). Often, the hardware description can also be compiled into a hardware model that can be emulated in a hardware emulator. A hardware emulator suitable for such use typically includes field programmable gate arrays (FPGAs) that serve as a breadboard for implementing the integrated circuit design. Both the simulation and the emulator typically run at a slower speed than a computer network (e.g., an Ethernet network).

When an integrated circuit that has a computer network interface is simulated or emulated, network activities are usually simulated or emulated at the speed of the circuit emulator or the circuit simulation. When using a circuit emulator, a conventional network-emulation device is typically connected to a port of the circuit emulator. The circuit emulator receives data packets from the network-emulation device, re-packages the data and transmits the re-packaged data back at the speed of the circuit emulator. The re-packaged data is then received by the network-emulation device, which inspects the re-packaged data to determine if the integrated circuit under emulation in the circuit

emulator correctly sends and receives data packets. However, on balance, such a conventional network-emulation device does not emulate network behavior accurately and correctly.

Alternatively, another conventional technique for connecting a circuit emulator to the network requires slowing down the network, receiving signals from the slowed network and translating the signals into suitable electrical signals in the form that the circuit emulator can accept. The circuit emulator, which typically operates at a slower speed than the network, can also send packets to the slowed network. However, because the network is designed to operate at a different speed, timing issues may arise in such a slowed network. These timing issues may require further modification to the network to resolve. Such modifications are undesirable because the modified network may not adequately represent network characteristics. Because not all network devices can be slowed to the circuit emulator speed, the circuit emulator is typically also limited to communication with a small subset of devices on the network.

Shortcomings of an emulation are typically also present in a logic circuit simulation.

SUMMARY OF THE INVENTION

The present invention allows a logic circuit simulator ("circuit simulation") running on a host processor to connect to a computer network at full network speed. The present invention provides a method and an apparatus for transferring data packets between a circuit simulation and the network. In one embodiment, an interface software program also installed on a host computer (e.g., a personal computer) is provided to handle communication between the network and the circuit simulator. The network can be, for example, an Ethernet network.

According to the present invention, data packets addressed to a device under simulation, or alternatively, addressed to a workstation connected to the network through the circuit simulation, is received and stored in buffers of the host computer. (In one example, a workstation is connected to a network through a simulated network interface.) The interface software in the host computer repackages the data packet into a second format for transmission to the simulated device at the speed of the circuit simulation. Under this arrangement, the interface software in the host computer need not send to the circuit simulation, for example, the preamble required to synchronize the clocks of the network and the circuit simulation, because the circuit simulation is usually not capable of providing an analog interface required to respond to the preamble. Similarly, the

interface software in the host computer repackages the data packets received from the circuit simulation into proper format for transmission to the network at full network speed. Under this arrangement, the existing memory in the host computer is used to buffer data packets communicated between the circuit simulation and the network, so that data packets received from the network at network speed are transmitted to the circuit simulation at a slower speed, and data packets received from the circuit simulation at the slower speed is provided to the network at full network speed.

In one embodiment, the present invention allows the interface software of a host computer to individually examine a data packet of a conventional off-the-shelf interface card to identify the beginning and the end of the packet. When the beginning and the end of a data packet can be identified, the interface software of the host computer ignores data packets not addressed to the simulated circuit. Consequently, compared to the prior art, a much smaller amount of buffer memory is required. This arrangement loses data packets only in the occasional event of a buffer overflow, thus effectively preventing network connection loss.

In one embodiment, the interface software of the host computer is implemented as a multithreaded program having, in one instance, two executing threads. One thread is a task that receives data packets from the network interface card, stores the received data packets in a buffer, retrieves the stored data for repackaging, and sends the repackaged data over a simulation interface to the circuit simulation. Another thread is a task that receives packets from the simulation interface, repackages the data into a network data packet and sends the network data packet over the network interface card to the network.

In another embodiment, the interface software of the host computer is implemented as a multithread program including four executing threads. One thread is a task that receives data packets from the network, and stores the received data packets in a buffer. A second thread is a task that polls the buffer for the received data packets. This second thread repackages the data packets and sends the repackaged packets over the simulation interface to the circuit simulation. A third thread is a task that receives data packets from the circuit simulation over the simulation interface and stores the received packets in a second buffer. A fourth thread is a task that polls the second buffer for the data packets received from the circuit simulation. This fourth thread repackages these data packets and sends the repackaged packets over the network interface to the network.

In yet another embodiment, the interface software of the host computer is also implemented as a multithread program, as in the previous embodiment, except that the

second buffer is eliminated and the third and fourth tasks are combined into a single task executing as a single thread. In this embodiment, the single task receives data packets from the simulation interface from the circuit simulation, repackages the data packets received and sends the repackaged packets over the network interface to the network. This approach is possible because a circuit simulation runs at a much slower speed than the network, such that a data packet received from the circuit simulation can be repackaged and sent to the network before the next data packet's arrival from the circuit simulation.

Further features and advantages of various embodiments of the invention are described in the detailed description below, which is given by way of example only.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 shows a configuration including first workstation 10, second workstation 20, host computer 30 and circuit emulator 12, in accordance with the present invention.

Figure 2 is a block diagram showing one configuration of circuit emulator 12, during an emulation of a network interface card.

Fig. 3 is a block diagram 300 showing the functions performed by Molasses program 50, in accordance with one embodiment of the present invention.

Fig. 4 shows a user interface Mainscreen 80 in Molasses program 50.

Fig. 5 shows a test setup suitable for a self-test in Molasses program 50, involving three personal computers (PC)

Fig. 6 is a block diagram 600 showing the functions performed by Molasses program 40, in accordance with a second embodiment of the present invention.

Figure 7 is a block diagram 700 showing the functions performed by Molasses program 750, in accordance with a third embodiment of the present invention.

Figure 8 is a block diagram 800 showing the functions performed by Molasses program 850, in accordance with a fourth embodiment of the present invention.

Figure 9 shows a configuration including first workstation 920 and host computer 930.

In the following detailed description, to simplify the description, like elements are provided like reference numerals.

DETAILED DESCRIPTION

The present invention is illustrated in the following using first, as an example, an emulation of a network interface device. At a latter part of this description, the present invention is illustrated, by way of another example, a simulation of a network interface device. Figure 1 shows a configuration including first workstation 10, second workstation 20, host computer 30 and circuit emulator 12. In this embodiment, a network interface card is emulated in circuit emulator 12, which interfaces with first workstation 10 over a conventional internal bus (e.g., a PCI bus). The network interface card emulated is intended to operate as a network interface of a workstation, such as workstation 10. However, circuit emulator 12 does not operate at the full network speed the network interface card is intended. Circuit emulator 12 is connected to host computer 30 over bidirectional interface 22, such as a conventional personal computer (PC) parallel port. Host computer 30 runs an interface program "Molasses" which is discussed in further detail below in conjunction with Fig. 4. Host computer 30 connects to conventional computer network 24 (e.g., 10BaseT Ethernet over twisted pair) using a conventional network interface. Workstation 20 communicates with host computer 30 over computer network 24 using conventional network protocols. Host computer 30 can be, for example, a desktop PC running Windows 95 or Windows 98 and equipped with a 10baseT Ethernet controller card and two parallel ports. A proprietary parallel port interface can be used for faster transfer speeds or easier connection to circuit emulator 12. In one embodiment, host computer 30 includes an Intel Pentium class processor and is equipped with 32 Mbytes of DRAM and 500 Mbytes of hard disk space. Host computer 30 also includes Media Access Controller ("MAC") drivers, parallel port drivers and the NDIS API (Network Driver Interface Specification - Application Program Interface). MAC drivers interface the operating system to the Ethernet card hardware. The parallel port drivers allow the operating system to interact with the parallel port hardware, and the NDIS API allows functional enhancements to network drivers in the operating system. Host computer 30 also includes a Graphical Users Interface ("GUI") and a packet capture, buffering and transmission application program as part of the "Molasses" program, which will be described in further detail later. Workstations 10 and 20 can each be any conventional workstation, including a PC running the Windows 98 operating system.

Alternatively, the network interface card can also be simulated in a circuit simulation running on host computer 30. Host computer 30 also runs a version of the interface program "Molasses" (e.g., Molasses program 750 or 850) which is discussed in further detail below in conjunction with Fig. 7 or Fig. 8.

Figure 2 is a block diagram of one configuration of circuit emulator 12 during an emulation of the network interface card. As shown in Figure 2, circuit emulator 12 provides logic circuit 18 that couples circuit emulator 12's circuit to bi-directional interface 22, an emulated Ethernet MAC 26, interface 23 to internal bus 21, and logic circuit 27, which is the remainder of the emulated network interface card. Cable assemblies 19A and 19B connect the input terminals and the output terminals ("I/O terminals") of Ethernet MAC 26 and logic circuit 18. Logic circuit 18 translates the signals of Ethernet MAC 26 (communicated over cable assemblies 19A) into the signals of bi-directional interface 22 for transmitting to host computer 30. In one embodiment, computer network 24 includes a conventional hub providing 10baseT connections. Alternatively, computer network 24 can also include a switch, which selectively transmits data packet based on destination addresses. Use of a switch can reduce packet traffic at a particular connection, and thus reduce the buffer requirements at the connected devices (e.g., host computer 30). Providing a switch at host computer 30's connection to computer network 24 also simplifies the Molasses program 50 running on host computer 30. In this configuration, emulator 12's connection to workstation 10 over internal bus 21 allows examination of signals in logic circuits 18 and 27 for debugging purpose.

Fig. 3 is a block diagram 300 showing the functions performed by Molasses program 50, in accordance with one embodiment of the present invention. Molasses program 50 includes a graphical user interface illustrated in Figure 4 by Mainscreen 80. As shown in Fig. 4, Mainscreen 80 allows the user to specify an Ethernet NIC (act 82) and a port address for bi-directional interface 22 connected to emulator 12 (act 84). Status line 92 displays continuously information about packets being processed by Molasses program 50. The amount of information to be shown on status line 92 can be selected using verbose mode option 86 and silent mode option 88. The user can also specify a log file (act 90) to record information such as the value of each byte in each data packet, a count or the nature of errors that occur, or comments that the user may wish to add. The log file can be used for future reference and debugging purposes.

Referring back to Figure 3, Molasses program 50 interfaces with network interface card (NIC) 74, which provides host computer 30's access to computer network 24, and interface 72, which couples host computer 30 through bi-directional interface 22

to circuit emulator 12. In this embodiment, interface 72 can be a conventional parallel port operating under the conventional EPP standard. Once the parameters of Mainscreen 80 are set, Mainscreen 80 calls "W32N_MolassesStart" routine 52. Routine 52 creates simultaneous threads running "W32N_MolassesBuffer" routine 54 and "PORT32_MolassesBuffer" routine 56, respectively. W32N_MolassesBuffer routine 54 receives packets from the Ethernet NIC 74 (via a W32N_PacketRead routine 58), stores the received packets into receive buffer 60 ("RPacketPack0") in host computer 30's main memory. Subsequently, the received packets in buffer 60 are transferred to transmit buffer 62 ("XPacketPack1"), from which they are then transmitted to interface 72 via "PORT32_PacketSend" routine 64. PORT32_MolassesBuffer routine 56 receives packets from interface 72 (via "PORT32_PacketRead" routine 66), stores the packets into receive buffer 68 ("RPacketPack1"). Subsequently, routine 56 then transfers the data packets in buffer 68 to a transmit buffer 70 ("XPacketPack0"), which are then transmitted to the Ethernet NIC 74 via "W32N_PacketSend" routine 76. Molasses program 50 converts data packet formats, when necessary. For example, the preamble that is used in a packet for synchronizing the clock signals of the network and the emulated device is removed before being forwarded to circuit emulator 12 over interface 72.

Mainscreen 80 calls "W32N_MolassesStop" routine 94 to terminate execution of both threads 54 and 56.

Figure 6 is a block diagram 600 showing the functions performed by Molasses program 40, in accordance with a second embodiment of the present invention. As in Molasses program 50 of Figure 3, Molasses program 40 of Figure 6 interfaces with network interface 74, which provides host computer 30's access to computer network 24, and interface 72, which couples host computer 30 to bi-directional interface 22 to circuit emulator 12. Interface 72 can be implemented by a conventional parallel port operating under, for example, the EPP standard. Once the parameters of Mainscreen 80 (Figure 4) are set, Mainscreen 80 calls "W32N_MolassesStart" routine 52, which creates four threads 120, 122, 124 and 126. Thread 120 executes "W32_PacketRead" routine 58, which receives data packets from Ethernet NIC 74 and stores the received data packet into shared buffer 128 in the main memory of the host computer 30. Thread 122 executes "Port32_PacketSend" routine 64, which polls shared buffer 128 for the received data packets, repackages these data packets and sends them to circuit emulator 12 over emulation interface (parallel port) 72. Thread 124 executes "Port32_PacketRead" routine 66, which receives data packets from circuit emulator 12 over parallel port 72 and stores the received data packet into shared buffer 130. Thread 126 executes a

"W32N_PacketSend" routine 76, which polls shared buffer 130 for data packets, repackages the data packets and sends them into network 24 over Ethernet NIC 74.

Because circuit emulator 12 typically runs at a speed much slower than devices on network 24, an alternative embodiment combines threads 124 and 126 and eliminates shared buffer 130, taking advantage that W32N_PacketSend routine 76 can complete repackaging and sending out a data packet to network 24 before arrival of the next data packet from circuit emulator 12 over parallel port 72.

Mainscreen 80 calls "W32N_MolassesStop" routine 94 to terminate execution of both threads 120, 122, 124 and 126.

The size of each of buffers 60, 62, 68 and 70, 128 and 130 can be changed dynamically. Even then, a buffer overflow condition can occasionally occur, resulting in data packets being discarded. Typically, discarding an incomplete packet risks losing a network connection. However, there is no risk of losing a network connection under the present invention, because only whole packets are discarded.

As mentioned above, the present invention is also applicable to a circuit simulation of the network interface device. Figure 9 shows a configuration including first workstation 920 and host computer 930. In this embodiment, a network interface card is simulated in a circuit simulation program running on host computer 930. Circuit simulation does not operate at the full network speed the network interface card is intended. Host computer 930 runs the interface program "Molasses" concurrently with the circuit simulation program. Host computer 930 connects to conventional computer network 924 (e.g., 10BaseT Ethernet over twisted pair) using a conventional network interface. Workstation 920 communicates with host computer 930 over computer network 924 using conventional network protocols. Host computer 930 can be, for example, a desktop PC running Windows 95 or Windows 98 and equipped with a 10baseT Ethernet controller card. In one embodiment, host computer 930 includes an Intel Pentium class processor and is equipped with 32 Mbytes of DRAM and 500 Mbytes of hard disk space. Host computer 930 also includes Media Access Controller ("MAC") drivers, parallel port drivers and the NDIS API (Network Driver Interface Specification - Application Program Interface). MAC drivers interface the operating system to the Ethernet card hardware. The NDIS API allows functional enhancements to network drivers in the operating system. A programming language interface (PLI) supported by the circuit simulation program allows the circuit simulation program to interact with the

Molasses software. Workstation 920 can each be any conventional workstation, including a PC running the Windows 98 operating system.

Fig. 7 is a block diagram 700 showing the functions performed by Molasses program 750, in accordance with a third embodiment of the present invention that can be implemented in the configuration of Figure 9 Molasses program 750 can include a graphical user interface similar to that illustrated in Figure 4 by Mainscreen 80. As in Molasses program 50, Molasses program 750 interfaces with network interface card (NIC) 74, which provides host computer 930's access to computer network 24, and simulation interface 772, which provides a program interface in host computer 930 to a circuit simulation running on host computer 930 concurrently with Molasses program 50. In this embodiment, simulation interface 772 can be a conventional programming language interface (PLI), an interprocess communication mechanism (e.g., a socket), a client-server type interface, or any other suitable software interface. For example, hardware description languages, such as Verilog or VHDL, provide support for programming interfaces. In particular, under Verilog, which is standardized by the Institute of Electrical and Electronics Engineers (IEEE), defines a PLI interface through which a compiled C Language program can communicate with a running circuit simulation.

Once the parameters of Mainscreen 80 are set, Mainscreen 80 calls "W32N_MolassesStart" routine 752. Routine 752 creates simultaneous threads running "W32N_MolassesBuffer" routine 754 and "SIM_MolassesBuffer" routine 756, respectively. As in W32N_MolassesBuffer routine 54 discussed above, W32N_MolassesBuffer routine 754 receives packets from the Ethernet NIC 74 (via the same W32N_PacketRead routine 58 discussed above), stores the received packets into receive buffer 60 in host computer 930's main memory. Subsequently, the received packets in buffer 60 are transferred to transmit buffer 62, from which they are then transmitted to simulation PLI 772 via "SIM_PacketSend" routine 764. SIM_MolassesBuffer routine 756 receives packets from simulation PLI 772 (via "SIM_PacketRead" routine 766), stores the packets into receive buffer 68. Subsequently, routine 56 then transfers the data packets in buffer 68 to a transmit buffer 70, which are then transmitted to the Ethernet NIC 74 via "W32N_PacketSend" routine 76. As in Molasses program 50 above, Molasses program 750 converts data packet formats, when necessary. For example, the preamble that is used in a packet for synchronizing the clock signals of the network and the emulated device is removed before being forwarded to the circuit simulation over simulation PLI 772.

Mainscreen 80 calls "W32N_MolassesStop" routine 794 to terminate execution of both threads 754 and 756.

Figure 8 is a block diagram 800 showing the functions performed by Molasses program 840, in accordance with a fourth embodiment of the present invention that can also be implemented in the configuration of Figure 9. As in Molasses program 750 of Figure 7, Molasses program 840 of Figure 8 interfaces with network interface 74, which provides host computer 930's access to computer network 24, and simulation PLI 772, which couples host computer 930 to the running circuit simulation. Once the parameters of Mainscreen 80 (Figure 4) are set, Mainscreen 80 calls "W32N_MolassesStart" routine 852, which creates four threads 820, 822, 824 and 826. Thread 820 executes "W32_PacketRead" routine 58, which receives data packets from Ethernet NIC 74 and stores the received data packet into shared buffer 128 in the main memory of the host computer 930. Thread 822 executes "SIM_PacketSend" routine 864, which polls shared buffer 128 for the received data packets, repackages these data packets and sends them to the running circuit simulation over simulation PLI 772. Thread 824 executes "SIM_PacketRead" routine 866, which receives data packets from the running circuit simulation over simulation PLI 772 and stores the received data packet into shared buffer 130. Thread 826 executes a "W32N_PacketSend" routine 76, which polls shared buffer 130 for data packets, repackages the data packets and sends them into network 24 over Ethernet NIC 74.

As in circuit emulator 12, because a circuit simulation typically runs at a speed much slower than devices on network 24, an alternative embodiment combines threads 824 and 826 and eliminates shared buffer 130, taking advantage that W32N_PacketSend routine 76 can complete repackaging and sending out a data packet to network 24 before arrival of the next data packet from the circuit simulation emulator 12 over simulation PLI 772.

Mainscreen 80 calls "W32N_MolassesStop" routine 894 to terminate execution of threads 820, 822, 824 and 826.

Molasses programs 50, 40, 750 and 840 each include a test program for self-test. Fig. 5 shows a test setup suitable for use with the self-test involving PCs 501, 502 and 503. For brevity, this test program is described with respect to Molasses program 50. Description herein regarding Molasses program 50 is equally applicable to Molasses program 40. The test program has two modes of operation - "initiate" and "respond". PC 502 runs Molasses program 50, configured to address two Ethernet network interface

cards, rather than the bi-directional interface, as in Figure 1. PC 501 runs the test software in initiate mode, generating and sending Ethernet packets of varying size to PC 502 via local area network 504 (e.g., Ethernet with 10BaseT connections) at step 521. At step 522, Molasses program 50 of PC 502 receives the packets from PC 501 using one of its two network interface cards, and then forwards the received packets to PC 503 over network 505, which is coupled to the other one of its network interface cards. PC 503 runs the test software in respond mode, taking each packet received from PC 502 (step 523) and re-transmitting it back to PC 502 over local area network 505 (step 524). At step 525, Molasses software of PC 502 then forwards the received packet from PC 503 to PC 501 over local area network 504. There, at step 526, under initiate mode, the test software on PC 501 compares the returned packet to the packet it transmitted at step 521. Any mismatch of these packets is reported as an error. In one embodiment, a timer can be set in 1/18-second increments to specify the frequency of packet generation in PC 102.

A user interface is provided by the test program to self-test Molasses program 50. The user interface displays an appropriate amount of information, based on a user's selection of silent mode or verbose mode. Through this user interface, a user can vary a packet throughput rate, effectuate an overflow condition in any of buffers 60, 62, 68, 70, 128 and 130, or test for timing and throughput problems. A status line is provided in the user interface to continuously update information about packets processed by Molasses program 50, such as the number of packets sent and received, the current value of the timer, an error count, and other status information desired. In addition, a log file can be specified to record the value of each byte of each packet, errors that occurred, and comments that the user may wish to add using a comment line in the user interface. The recorded information may be used for future reference and debugging.

The test program also provides a test for accessing circuit emulator 12 through a bi-directional interface (e.g., a parallel port). In one embodiment, an industry standard parallel port conforming to the enhanced parallel port (EPP) standard is provided. The test program allows a user to read and write 8-bit addresses and 8-bit data patterns via the parallel port to circuit emulator 12. In one test, data is continuously written to and read back from circuit emulator 12 and compared. Any mismatch between the written data and the read back data is reported as an error.

Various modifications and adaptations of the operations described here would be apparent to those skilled in the art based on the above disclosure. Many variations and modifications within the scope of the present invention are therefore possible. The present invention is set forth by the following claims.